título

Procesamiento masivo de datos mediante Cassandra y Spark

Máster en Sistemas Informáticos Avanzados
Septiembre de 2016

Autor:

Xabier Zabala Barandiaran

Supervisores:

German Rigau i Claramunt
UPV/EHU
Iñigo Etxabe
Datik Información Inteligente S.L.

Agradecimientos

En primer lugar, quisiera expresar mi gratitud a las personas que han posibilitado la concepción y el desarrollo de la presente tesina. Agradezco a German Rigau i Claramunt, director del proyecto en la UPV/EHU, el asesoramiento ofrecido durante el transcurso del mismo. Doy también las gracias a Iñigo Etxabe, cofundador y CTO de Datik Información Inteligente, por facilitar los medios tecnológicos necesarios para llevar a cabo el trabajo y depositar total confianza en mi labor desde el primer día.

Agradecer, cómo no, a mis padres José Javier Zabala y María Pilar Barandiaran el esfuerzo desempeñado para allanar, en la medida que les ha sido posible, el camino que he recorrido hasta el día de hoy. Me congratula haber sabido responder satisfactoriamente a la confianza que han depositado en mí. Por lo dicho y por todo lo que suponen para mí, un beso enorme para los dos.

No me quisiera olvidar de aquellas personas que me han acompañado durante este maravilloso periplo. Doy las gracias a los amigos de siempre por el apoyo incondicional mostrado desde la distancia y a la gente que he tenido la fortuna de conocer durante los años de universidad. Quisiera agradecer especialmente a las personas que en este breve periodo se han ganado a pulso el privilegio a ser parte importante de lo que me resta de existencia, a los cuales me atrevo a mencionar aún con el temor de dejarme a alguna en el tintero: Adrián Núñez, Eider Irigoyen, Jaime Altuna y Marta García. No hallo palabras para expresar la gratitud que siento hacia vosotras.

Por último, pero no por ello menos importante, quisiera evocar a todos los docentes que han tomado parte en mi formación desde aquel Septiembre del 2008 y agradecer a todos ellos el conocimiento inculcado y el esfuerzo invertido en mi persona durante estos años.

Gracias de todo corazón a la gente mencionada en este breve capítulo por haber hecho de mí un mejor profesional y una mejor persona, además de darme las fuerzas necesarias para seguir evolucionando en ambos aspectos de cara al futuro.

Resumen

Proyecto Final del Máster en Sistemas Informáticos Avanzados. Estudio empírico sobre el rendimiento ofrecido por varias tecnologías emergentes en el campo del Big Data en comparación a una base de datos tradicional a la hora de operar en escenarios que requieren un almacenamiento y procesamiento eficaz de volúmenes masivos de datos.

Dos entornos de pruebas totalmente aislados han sido erigidos sobre la misma máquina física. En el primero, se ha construido un clúster compuesto por tres nodos virtuales que operan en la misma red privada. Dichos nodos han sido dotados de tecnología necesaria para el correcto funcionamiento de Apache Cassandra [5] y Apache Spark [12]. Sobre el segundo entorno, constituido por un nodo virtual de potencia equivalente al clúster ya mencionado, se ha instalado una instancia de MySQL Server. Una vez habiendo diseñado un conjunto de consultas equivalentes para ambas bases de datos, se ha procedido a poblar dichos sistemas de almacenamiento utilizando un data-set público de aproximadamente 25GB. Para finalizar, las consultas predefinidas han sido ejecutadas pudiendo así cuantificar el tiempo de respuesta ofrecido por cada entorno.

El estudio evidencia que a la hora de trabajar con volúmenes masivos de datos el binomio formado por Apache Cassandra y Apache Spark, además de ofrecer una solución totalmente escalable y tolerante a fallos, mejora sustancialmente el rendimiento ofrecido por MySQL Server. No obstante, para gozar de dichas ventajas, se antoja necesario invertir más tiempo en analizar exhaustivamente la naturaleza de los datos que se desean tratar.

Palabras Clave: Apache Cassandra, Apache Spark, Benchmark, Big Data, MySQL Server.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Desde Aristóteles y su libro Segundos Analíticos [1] hasta Galileo, padre de la ciencia moderna, adalides del conocimiento han proclamado que un método de investigación basado en lo empírico y en la medición, sujeto a los principios específicos de las pruebas de razonamiento es el camino para conocer la verdad.

Hoy en día, época en la que los avances tecnológicos han posibilitado observar y medir de forma exhaustiva un gran abanico de fenómenos, la ingente cantidad de datos que se genera en el proceso es, a veces, intratable mediante las tecnologías convencionales, y por ende, es imposible extraer todo el conocimiento que atesoran. El problema, lejos de atenuarse, se acrecienta con el paso del tiempo. Estudios como el realizado por McKinsey Global Institute [2] estiman que el volumen de datos que se genera crece un 40 % cada año y auguran que entre 2009 y 2020 se verá multiplicado por 44 [7].

Para lidiar con dicha problemática, en los últimos años ha irrumpido la necesidad de desarrollar nuevas metodologías y tecnologías que permitan operar eficientemente sobre masas colosales de datos, dando como resultado el nacimiento del Big Data [6].

Empresas de renombre mundial, conscientes de los beneficios que el Big Data les puede reportar en diferentes facetas de su modelo de negocio, ya se han interesado en este fenómeno. De un estudio realizado entre los altos ejecutivos de las firmas que lideran el Wall Street se desprende que el 96 % tiene planeadas ciertas iniciativas relacionadas con el Big Data, y el 80 % ya tiene finalizada alguna [10].

[1] Órganon II de Aristóteles: Recopilación de obras Aristotélicas que incluye el libro Segundo Analíticos
[2] McKinsey Global Institute

1.1. Contexto

Datik Información Inteligente [3] es una empresa tecnológica perteneciente al Grupo Irizar [4] que desarrolla soluciones ITS destinadas a la gestión del trasporte, tanto ferroviario como por carretera y movilidad ciudadana.

Uno de los productos estrella de la entidad es el denominado iPanel, concentrador de información que ofrece al operador de transporte servicios de valor añadido en la gestión de la información generada por su flota. El funcionamiento del servicio se puede resumir mediante la Figura 5:

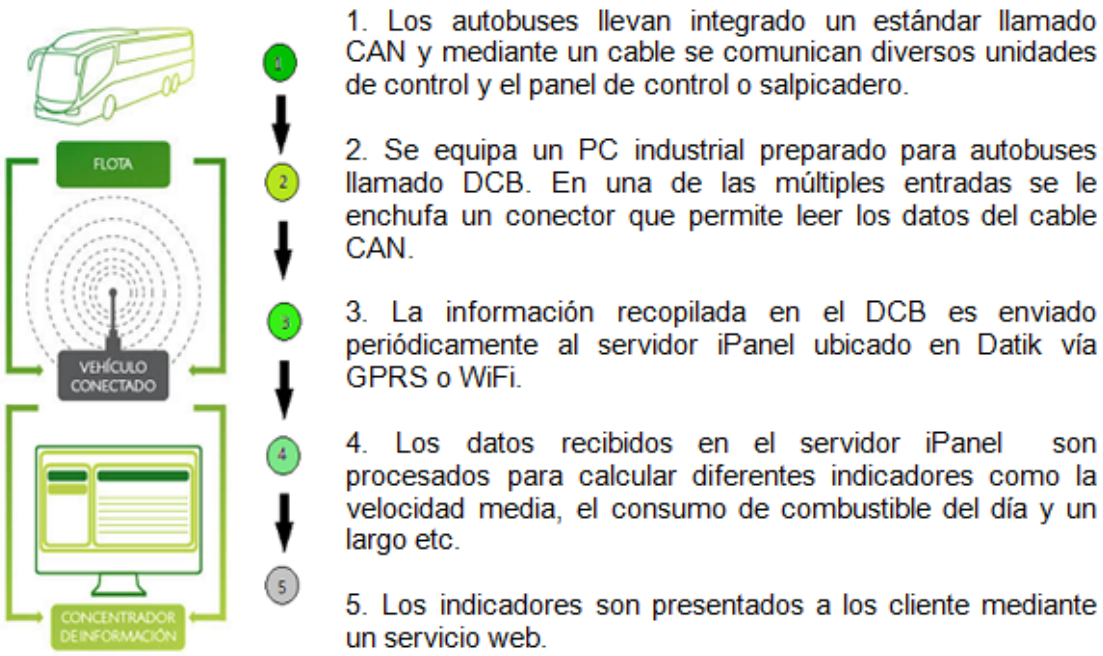

Figura 1: Funcionamiento resumido de iPanel

La incesante integración de nuevos vehículos a iPanel ha generado un crecimiento exponencial en el número de registros almacenados en ciertas tablas MySQL. Aunque el volumen actual no llega a suponer riesgo alguno para el funcionamiento del servicio, Datik tiene identificados varios escenarios en los que la situación se podría revertir, causando graves inconvenientes.

[3] http://www.datik.es/
[4] http://www.irizar.com/irizar/

Fiel reflejo de ello es el Cálculo de Indicadores: proceso de ejecución diaria que realiza operaciones aritméticas intensivas sobre datos almacenados en diversas tablas para después, agrupar los resultados en base a diferentes criterios. Siendo dichas tablas las que mayor crecimiento experimentan, el aumento del volumen de las mismas incrementa de forma desorbitada el tiempo necesario para finalizar el cálculo, pudiendo, en un futuro, llegar a tardar más de 24 horas y cancelar los indicadores que ofrecen la información del último día.

El objetivo del presente proyecto es proponer soluciones tecnológicas que solvente permanentemente el problema del proceso Cálculo de Indicadores y que a su vez, pueda valer para lidiar con otros obstáculos de la misma índole que pudieran emerger en un futuro.

1.2. Propuesta

Tal y como se ha mencionado en el apartado anterior, la problemática que envuelve al Cálculo de Indicadores es originado por el incremento exponencial de datos que dicho proceso ha de tratar. No sería descabellado pensar que la solución pudiese pasar por escalar verticalmente la máquina y afinar la configuración de MySQL. No obstante, ambas mejoras son limitadas, mientras que el volumen de los datos seguirá en aumento de forma inexorable, volviendo, tarde o temprano, a tener que lidiar con los mismos problemas del principio.

Siendo imposible reconducir la situación mediante las tecnologías tradicionales, en el presente proyecto se propone realizar un cambio de paradigma que implica migrar las tablas relacionadas con los Indicadores a un modelo distribuido que posibilite escalar la infraestructura horizontalmente. A su vez, se sugiere dividir el problema en dos apartados, almacenamiento y procesamiento, dotando la infraestructura de tecnología adecuada para ofrecer una respuesta eficaz a cada una de las partes.

En cuanto a almacenamiento se refiere, debido a la imperante necesidad de escarbar en hercúleos volúmenes de datos, se propone utilizar una base de datos no relacional. Dentro de la extensa y heterogénea gama de sistemas de almacenamiento NoSQL [5] existentes a día de hoy, se ha considerado idóneo el uso de Apache Cassandra [5], una base de datos distribuida del tipo clave-valor que ofrece alta disponibilidad sin un solo punto de fallo, un ratio de escrituras por segundo sustancialmente superior en comparación a sus homólogos [8] y velocidades de lectura nada desdeñables.

[5] `http://nosql-database.org/`

Por su parte, la tecnología seleccionada para el procesamiento ha sido Apache Spark [12]. Se trata de una plataforma de computación en clúster que ofrece una interfaz para programar operaciones paralelizadas y tolerantes a fallo que ha irrumpido con fuerza en el último lustro. A logrado desbancar tecnologías semejantes como MapReduce [2] gracias a la capacidad de almacenar en memoria los resultados intermedios del cómputo posibilitando así acelerar el procesamiento hasta 100 veces [11] en determinados escenarios.

Para cuantificar los beneficios aportados por las tecnologías propuestas se ha decidido utilizar un Dataset público de aproximadamente 25 GB que recoge la información de los trayectos realizados por los taxis amarillos de Nueva York durante el año 2015, ya que los datos actualmente almacenados por Datik se encuentran sujetos a una clausula de confidencialidad.[6]

1.3. Organización del documento

El presente documento abarca el estudio empírico realizado sobre el rendimiento obtenido mediante el uso conjunto de Apache Cassandra y Apache Spark en comparación al ofrecido por MySQL Server en escenarios que requieren un almacenamiento y procesamiento eficaz de volúmenes masivos de datos.

En este primer capítulo, se ha introducido la problemática que ha impulsado la creación del proyecto y expuesto la propuesta tecnológica para dar solución a la misma.

En el capítulo 2, se desgrana el funcionamiento de las tecnologías electas para resolver el problema, haciendo especial hincapié en las peculiaridades que les permiten ofrecer una mejora sustancial en comparación a las herramientas tradicionales.

Una vez en el capítulo 3, se detalla la praxis llevada a cabo para confeccionar los dos entornos de prueba. Por un lado, se describe la construcción de un entorno centralizado compuesto por un único nodo que alberga una instancia de MySQL y por el otro, la de uno distribuido, formado por tres nodos homogéneos sobre el cual operan Apache Cassandra y Apache Spark en consonancia.

El almacenamiento y la retribución de los datos constituyen el grueso del 4to capítulo. Primero, se expone la naturaleza de los datos que conforman el data-set y se enuncian las

[6] `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`

consultas que se desean realizar sobre este. Después, se presenta el diseño de las tablas utilizadas para almacenar dichos datos, incidiendo especialmente en la correspondiente a Apache Cassandra debido a las peculiaridades que presenta. Para finalizar, se detalla el preproceso realizado sobre los datos para almacenarlos de forma equivalente en ambas bases de datos.

En el capítulo 5, se publican los resultados obtenidos una vez habiendo ejecutado las consultas anteriormente definidas. Dichos resultados son representados mediante gráficas y analizados posteriormente para observar de forma nítida las diferencias existentes entre ambos entornos en lo que a tiempo de procesamiento se refiere.

Para finalizar, en el capítulo 6 se presentan las conclusiones y líneas futuras para el proyecto.

Fuera de la estructura general de la memoria, se puede encontrar la bibliografía citando todas las fuentes relevantes accedidas en el devenir del proyecto.

Capítulo 2

Análisis de las tecnologías propuestas

2.1. Apache Cassandra

Apache Cassandra es una base de datos distribuida que permite operar sobre grandes volúmenes de datos del tipo clave/valor. Se caracteriza por ofrecer una disponibilidad total y mayor escalabilidad lineal en comparación a otras bases de datos NoSQL, utilizando, para ello, una serie de nodos homogéneos que se comunican mediante un protocolo P2P de replicación asíncrona, lo cual permite realizar operaciones de baja latencia para todos los clientes sin necesidad de un servidor maestro.

Fue concebida en el 2008 por los ingenieros de Facebook basándose en sistemas de almacenamientos distribuidos como Dynamo (Amazon) [3] y BigTable (Google) [1] con el propósito de mejorar la funcionalidad de búsqueda en la bandeja de entrada. En 2012, investigadores de la Universidad de Toronto que estudian los sistemas NoSQL concluyeron que .[En] términos de escalabilidad, hay un claro ganador a través de nuestros experimentos. Cassandra logra el mejor rendimiento para el número máximo de nodos en todos los experimentos"[8].

Actualmente Apache Cassandra es utilizada por infinidad de aplicaciones en negocios modernos, siendo la base de datos elegida por un tercio de las compañías que conforman la Fortune 100 [1]. Claro ejemplo de ello son empresas mundialmente conocidas como Apple, Facebook o NetFlix, los cuales utilizan Apache Cassandra como parte de su entramado tecnológico desde hace ya unos años. Cabe destacar que el uso del mismo no se limita al mundo empresarial. Muestra de ello es la acogida que ha tenido en el ámbito de la investigación, formando parte en experimentos punteros a nivel mundial como al-

[1] `http://fortune.com/2015/04/14/datastax-hp-sales-partnership/`

gunos de los realizados en el CERN [9].

2.1.1. Funcionamiento

Las bases de datos NoSQL, en su mayoría, han sido concebidas para lidiar con un conjunto especifico de problemas. Debido a ello, para entender el funcionamiento de Apache Cassandra se ha decidido resaltar las peculiaridades que le distinguen de los demás sistemas de almacenamiento no relacionales.

Base de datos distribuida de alta disponibilidad

El Teorema de Brewer [4], también conocido como Teorema CAP , enuncia que un sistema de cómputo distribuido no puede garantizar simultáneamente las tres propiedades que se presentan a continuación, solo pudiendo cumplir dos de ellas al mismo tiempo, y acabar cumpliendo la restante tarde o temprano:

- **Consistencia**(Consistency): Todos los nodos ven la misma información al mismo tiempo.
- **Disponibilidad**(Availability): La garantía de que cada petición a un nodo reciba una confirmación de si ha sido o no resuelta satisfactoriamente.
- **Tolerancia al Particionado**(Partition Tolerance): El sistema sigue funcionando a pesar de que haya sido partido por un fallo de red.

Para una base de datos distribuida que promete la disponibilidad completa, el Teorema de Brewer implica la incapacidad de garantizar la consistencia total de los datos que almacena y la necesidad de esperar un tiempo indeterminado para que las réplicas de un registro modificado se actualicen correctamente.

Para lidiar con dicha restricción, Cassandra ofrece la posibilidad de afinar el nivel de consistencia [2] en cada consulta sacrificando para ello parte de la disponibilidad. Las consultas de mayor disponibilidad responden con el dato ofrecido por la primera réplica en contestar sin siquiera cotejar la existencia de una versión actualizada del mismo, mientras que las consultas de mayor consistencia podrán fallar por la caída de un nodo que alberga una de las réplicas debido a no poder satisfacer el nivel de consistencia requerido. Por tanto, queda en manos del diseñador el trabajo de encontrar el equilibrio entre ambas características, siendo la naturaleza de los datos a tratar un factor determinante para ello.

[2] `https://docs.datastax.com/en/cassandra/2.1/cassandra/dml/dml_config_consistency_c.html`

Optimizado para escrituras

Una de las características a destacar de Cassandra es el elevado ratio de escrituras por segundo que es capaz de realizar, superando notoriamente a otras bases de datos NoSQL en este apartado [8].

Al ejecutar una consulta que desemboca en una escritura, los registros son almacenados en dos estructuras denominadas CommitLog y Memtable. El primero, es un fichero donde se adjuntan los registros recibidos, mientras que el segundo, es una cache en memoria que apila dichos registros hasta llegar a su capacidad máxima. Sólo cuando ambas estructuras finalizan de guardar los datos, Cassandra considera que la consulta ha sido ejecutado satisfactoriamente pudiendo pasar a procesar otras operaciones.

Cuando la Memtable se llena empieza un proceso de vaciado en el cual, primero, se dilucida el nodo destinatario de cada registro. Después, dichos registros son escritos de forma secuencial en estructuras inmutables denominadas SSTable, ficheros que representan físicamente el contenido de una tabla en Cassandra. Si esta operación es interrumpida, se utiliza la información almacenada en el CommitLog para recuperar las escrituras perdidas. Finalmente, cuando todos los registros han sido transferidos su correspondiente SSTable el CommitLog es purgado.

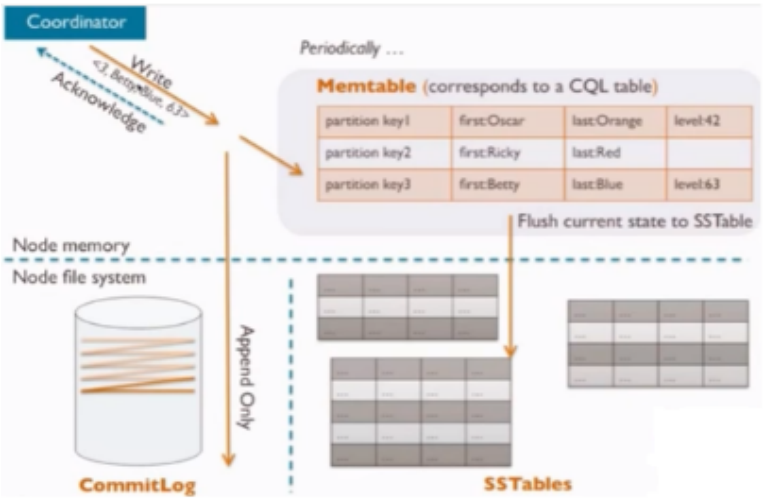

Figura 2: Almacenamiento de datos en Cassandra

Debido a la naturaleza inmutable de las SSTable, cada vez que la Memtable se vacía un nuevo fichero es generado para representar una misma tabla. Ello puede conllevar problemas de eficiencia a la hora de consultar la información de dicha tabla, ya que será totalmente necesario leer todos los ficheros correspondientes. Para sobreponerse a este inconveniente, Cassandra posee un mecanismo llamado compactación.

Replicación asíncrona sin maestro

//Gossip Protocol (paper?)

Protocolo de comunicación peer-to-peer que intercambia el estado del propio nodo y de aquellos cuyo estado conoce de forma periódica.

El proceso es ejecutado cada segundo e intercambia información con otros tres nodos del cluster. De esa forma, todos los nodos aprenden sobre el resto de manera rapida.

Cada mensaje Gossip tiene una versión asociada a él, gracias al cual el nodo que recibe dicho mensaje puede comparar con la información que tiene y sobreescribir su conocimiento en caso de ser necesario.

//Failure Detector (paper?)

Necesidad de diferenciar un nodo caído con otro en estado transitorio para redireccionar las peticiones solo a los nodos activos.

El detector de fallos es un método que permite determinar localmente si otro nodo del sistema esta caido o en funcionamiento mediante el uso de los estados gossip y el historial.

Cassandra utiliza accrual detection mechanism para calcular un umbral por nodo que tenga en cuenta el rendimiento de red, carga de trabajo, u otras condiciones.

Es posible configurar dicho umbral modificando el valor de la variable $\text{phi}_{c}onvict_{t}hresholdparaajustar$

// merkle tree (paper?)

Consiste en comparar todas las réplicas de cada dato que existe o debería existir y actualizarlos a la versión más reciente mediante el uso de Merkle Trees.

Un Merkle tree es un árbol de hash donde los nodos padres más altos en el árbol son los hashes de sus respectivos hijos. La ventaja principal de árbol de Merkle es que cada rama del árbol se puede comprobar de forma independiente , sin la necesidad de descargar todo el conjunto de datos.

La implementación de Anti-Entropia de Cassandra genera un Merkle tree por cada column family a la hora de la compactación y se almacenan solo hasta enviarlos a los nodos vecinos.

Implica enviar un exceso de datos por la red pero ahorra en operaciones I/O del disco local, lo cual es preferible para conjuntos de datos muy grandes.

Sin punto único de fallo

//replicacion factor + diapositivas

///

Analizando las características propias de Cassandra y en concreto su estructura, la Keyspace es un espacio de nombres para un conjunto de ColumFamily. Por lo general, se utiliza uno por aplicación y es considerado como el equivalente a una base de datos del modelo relacional. El ColumFamily, a su vez, es capaz de almacenar diferentes columnas, siendo el homólogo de una tabla del modelo relacional. Para finalizar, una columna está

compuesta por clave y valor, además de un campo del tipo timestamp gracias al cual se actualizan las réplicas obsoletas.

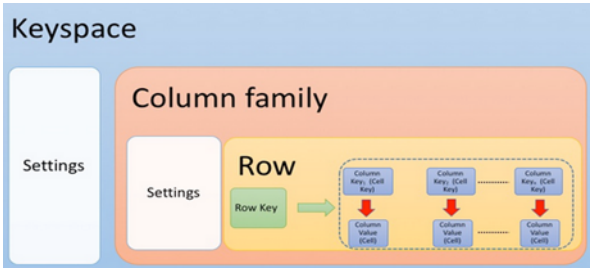

Figura 3: Estructura de Cassandra

A la hora de definir un keyspace dos atributos estrechamente relacionados han de ser especificados: Replication Factor y Replica Placement Strategy(). El primero, consiste en un número que indica cuantas copias de un mismo registro se han de almacenar en la infraestructura. Cada nodo posee una réplica para un cierto rango de datos y si uno de ellos falla, otro que posea dicha réplica puede responder a la petición sin tener que interrumpir el servicio. El segundo, define cómo se han de repartir los registros replicados por el anillo, ofreciendo distintas opciones dependiendo si los nodos del clúster se encuentran alojados en un único datacenter o repartidos entre varios.

El atributo Replica Placement Strategy cobra especial interés al operar sobre un clúster cuyos nodos están distribuidos en punto geográficos lejanos, ya que permite almacenar cada réplica en diferentes puntos del globo. De esa manera, se consigue minimizar la latencia de las peticiones recibidas desde diferentes puntos del plantea y además, proteger la base de datos de una posible caída a nivel de datacenter debido a diversos catástrofes.

Si un componente de Cassandra tiene especial importancia en su funcionamiento es el ColumFamily. En una base de datos relacional como MySQL, las tablas son diseñadas con el objetivo de minimizar la redundancia de los datos almacenados en ella, siguiendo para lograr dicho objetivo un proceso de normalización[]. En cambio, al operar con Cassandra, ocurre exactamente lo contrario. Las tablas se construyen buscando una respuesta rápida a las consultas sin importar que ello implique tener que almacenar datos redundante en la base de datos.

patition key etc
consecuencia: restriccion en las consultas que se pueden realizar

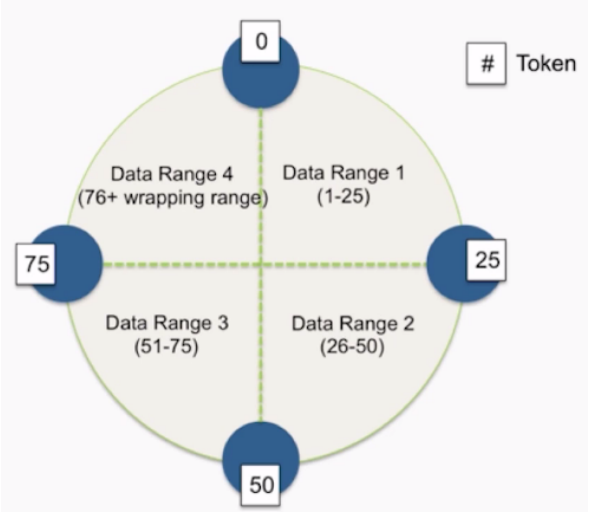

Figura 4: Particionamiento en Cassandra

2.1.2. Cassandra Query Lenguage (CQL)

permite conectarse al cualquier nodo del cluter (homogeneidad)

Apache Cassandra posee su propio lenguaje de consultas, el denominado Cassandra Query Lenguage (CQL). Su sintaxis guarda una gran similitud con la de SQL, lo cual facilita, de forma notoria, el salto que supone pasar de trabajar con bases de datos relacionales a distribuidos.

Aún siendo sintácticamente tan parecido a SQL, presenta ciertas restricciones debido a que es un lenguaje de consultas de una base de datos distribuida. Por ejemplo, no ofrece la posibilidad de realizar operaciones como JOIN y es totalmente necesario especificar todas los atributos que componen la clave primaria a la hora de realizar cualquier consulta de filtrado o de actualización en la tabla. La única operación que no cumple esta restricción es un select que contenga la clausula where, ya que, al definir la clave primaria Cassandra indexa de forma automática todos sus componente, posibilitando más tarde hacer uso de ellos en este caso concreto.

Otra de las peculiaridades que presenta CQL es el hecho de ofrecer dos modos distintos de realizar un update. El primero de todos es el mencionado en el párrafo anterior. El segundo posibilita actualizar una columna realizando un insert repitiendo el valor de las claves primarias de una columna ya existente en la base de datos. Esta segunda forma es cómoda a la par de peligrosa porque Cassandra no notifica si una clave primaria ya existe en la base de datos o no, pudiendo un insert desencadenar en un update no deseado.

2.2. Apache Spark

Apache Spark[9] es un proyecto open source de computación en clúster. Desde el principio fue diseñado para poder ejecutar algoritmos iterativos en memoria sin la necesidad de almacenar en disco los resultados intermedios generados durante el proceso. Esta peculiaridad permite que los procesamientos llevados a cabo con Spark puedan llegar a ser, en algunos casos concretos, 100 veces más rápidos que los de MapReduce[10].

A mediados de 2014, coincidiendo con el lanzamiento de la primera versión, alcanzó la cifra de 465 colaboradores, convirtiéndolo en el proyecto más activo entre los relacionados con el Big Data dentro de la Apache Software Fundation.

Apache Spark está compuesto por múltiples y variados componentes que pueden ser utilizados de forma conjunta.

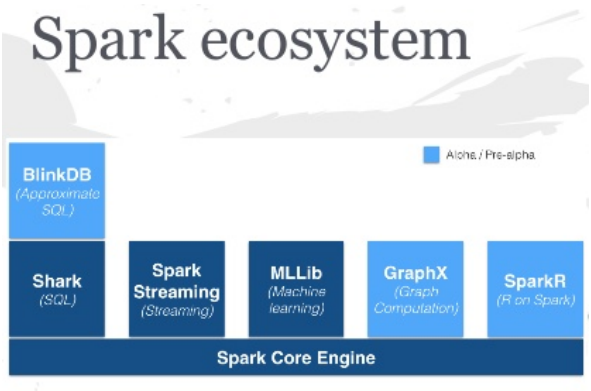

Figura 5: Ecosistema Spark

La base del proyecto es el denominado Spark Core. Proporciona envío distribuido de tareas, planificación y funciones básicas de entrada salida. La abstracción fundamental de programación se llama Resilient Distributed Datasets (RDD)[11], una colección lógica de datos particionados a través de las máquinas que se expone mediante una API integrada en lenguajes como Java, Python y Scala.

2.2.1. Funcionamiento

Para el funcionamiento de Spark, es condición sine qua non que los nodos de la infraestructura tengan acceso a la totalidad de los datos que se desea tratar. Ello implica que para procesar un fichero de 50GB, cada nodo tendría que poseer una copia del mismo

almacenado en su disco. Esta praxis es inviable, ya que más allá de los problemas de consistencia que generaría, para nada es eficiente ocupar la memoria de todos los nodos con información redundante y procesar el fichero entero cuando en realidad se va a hacer uso de una pequeña porción de dichos datos en cada ejecución.

Las bases de datos distribuidas como Cassandra solventan los problemas anteriormente mencionados. Se encargan de distribuir los datos entre diferentes nodos del clúster, ofrecen la posibilidad de acceder a ellos desde cualquier punto y mantienen la consistencia de los mismos a cambio de sufrir una pequeña latencia en el caso de requerir información almacenada en otro nodo de la infraestructura.

Al ejecutar una aplicación que opera con Spark, un componente denominado driver es lanzado. Debido a la necesidad de obtener recursos (CPU y memoria) para llevar a cabo la computación que se le ha encomendado, se comunica con un nodo del clúster que, mediante especificación previa, adopta el rol de maestro. Éste pregunta a todos los nodos que conforman la infraestructura sobre la cantidad de recursos disponibles que poseen y así asignarles los executors correspondiente. Las máquinas que alojen al menos un executor pasan a denominarse worker y a partir de este momento, cada executor podrá comunicarse directamente con el Driver para poder recibir las tareas que éste le envíe.

Un executor es una unidad de trabajo que se encarga de computar las tareas que le encomienda el driver. El número de executors que puede albergar cada worker está directamente relacionado con el número de procesadores que este posee. De la misma forma, es posible repartir la memoria RAM que dispone el nodo worker entre varios executors. Spark permite modificar ambos parámetros programáticamente permitiendo así poder amoldarse a las particularidades de cada ejecución.

Para transferir el código del programa, residente en la máquina del driver, éste adopta el rol de servidor e intenta enviar dicho código a los workers. Si el fichero JAR que contiene el código ha sido recibido correctamente por sus destinatarios, estos responden mediante un ACK y en caso contrario, se vuelve a intentar el envío un número determinado de veces. Una vez llegado al máximo de reintentos, el worker que no haya enviado el ACK es considerado como caído, quedando los executors que albergaba fuera del posterior reparto de tareas.

A la hora de realizar operaciones en Spark, el objeto estrella es el denominado Resilient Distributed Datasets (RDD)[11]. Se trata de una abstracción que mediante diferentes APIs disponibles para Java, Scala y Python permite manipular datos distribuidos por los diferentes nodos del clúster como si estuvieran almacenados de forma local. Este objeto es inmutable, lo cual implica que una vez creado no se le pueden añadir nuevos

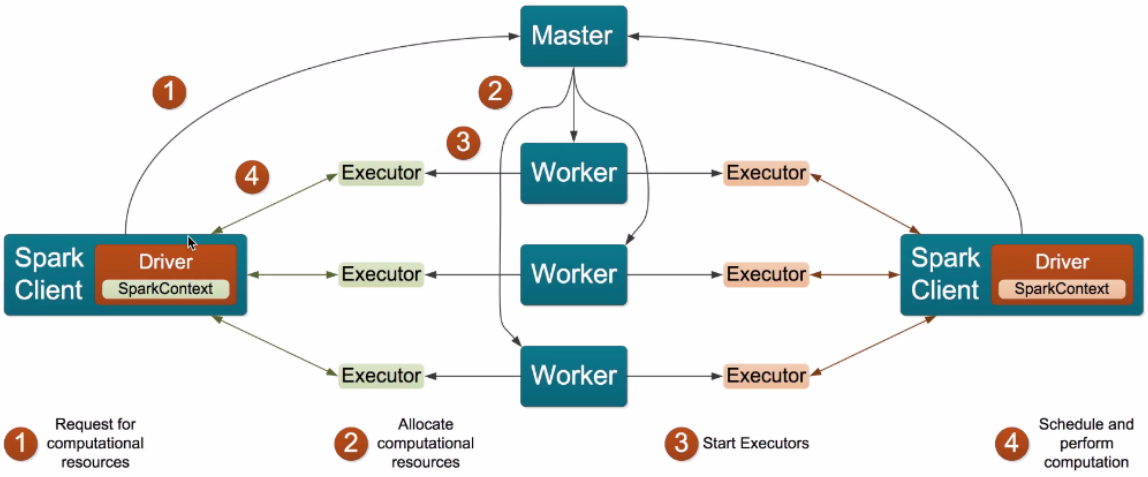

Figura 6: Arquitectura Spark

elementos o eliminar los existentes, solo aplicar transformaciones y acciones sobre el.

Las operaciones que se pueden realizar sobre las RDD se agrupan, tal y como se ha adelantado antes, por transformaciones y acciones. Las primeras transforman un RDD en otro según el criterio indicado y las segundas realizan modificaciones sobre los datos almacenados en dichas RDD. Cabe destacar que las transformaciones en Spark son operaciones "lazy", lo cual implica que en realidad cada nodo memoriza la secuencia de transformaciones que ha de realizar y los procesa cuando una acción es ejecutada.

Una vez terminada la primera fase en la que los executor son creados y enlazados con el driver, éste último empieza a analizar la estructura del código y genera un grafo DAG (Directed Acyclic Graph) con las operaciones que se realizan sobre la RDD. Partiendo de ese grafo genera un job por cada operación de tipo acción que encuentra y dentro de cada job separa la ejecución en diferentes stages según las dependencias que existan entre operaciones. Por último, cada stage es dividido por defecto en unidades de 64MB y a cada unidad resultante se denomina task, el cual es enviado a un executor para ser procesado. El tamaño de cada task puede ser modificado programáticamente, pudiendo de esa forma manipular el número de task que un executor deba ejecutar.

El driver, una vez habiendo recibido los resultados de todas los task que ha repartido, enviará un mensaje a los executors indicando que el procesamiento ha sido finalizado y calculará el resultado final ofreciendo la posibilidad de, por ejemplo, almacenarlo en una base de datos distribuida como Cassandra.

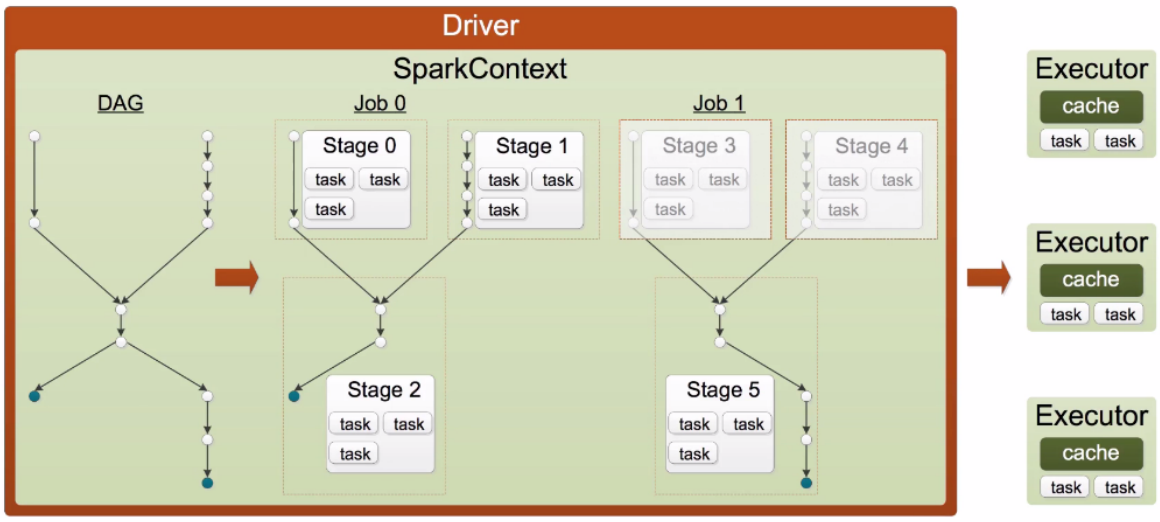

Figura 7: Arquitectura Spark

Bibliografía y Referencias

[1] Fay Chang y col. “Bigtable: A distributed storage system for structured data”. En: *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008), pág. 4.

[2] Jeffrey Dean y Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters”. En: *Communications of the ACM* 51.1 (2008), págs. 107-113.

[3] Giuseppe DeCandia y col. “Dynamo: amazon’s highly available key-value store”. En: *ACM SIGOPS Operating Systems Review* 41.6 (2007), págs. 205-220.

[4] Seth Gilbert y Nancy Lynch. “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services”. En: *ACM SIGACT News* 33.2 (2002), págs. 51-59.

[5] Avinash Lakshman y Prashant Malik. “Cassandra: a decentralized structured storage system”. En: *ACM SIGOPS Operating Systems Review* 44.2 (2010), págs. 35-40.

[6] James Manyika y col. “Big data: The next frontier for innovation, competition, and productivity”. En: (2011).

[7] Raghunath Nambiar. “Towards an Industry Standard for Benchmarking Big Data Workloads”. En: ().

[8] Tilmann Rabl y col. “Solving big data challenges for enterprise application performance management”. En: *Proceedings of the VLDB Endowment* 5.12 (2012), págs. 1724-1735.

[9] Alexandru Dan SICOE y col. *A Persistent Back-End for the ATLAS Online Information Service (P-BEAST)*. Inf. téc. ATL-COM-DAQ-2012-010, 2012.

[10] *The State of Big Data in the Large Corporate World*. Big Data Executive Survey 2013 Boston. 2013.

[11] Reynold S Xin y col. “Shark: SQL and rich analytics at scale”. En: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*. ACM. 2013, págs. 13-24.

[12] Matei Zaharia y col. “Spark: cluster computing with working sets.” En: *HotCloud* 10 (2010), págs. 10-10.